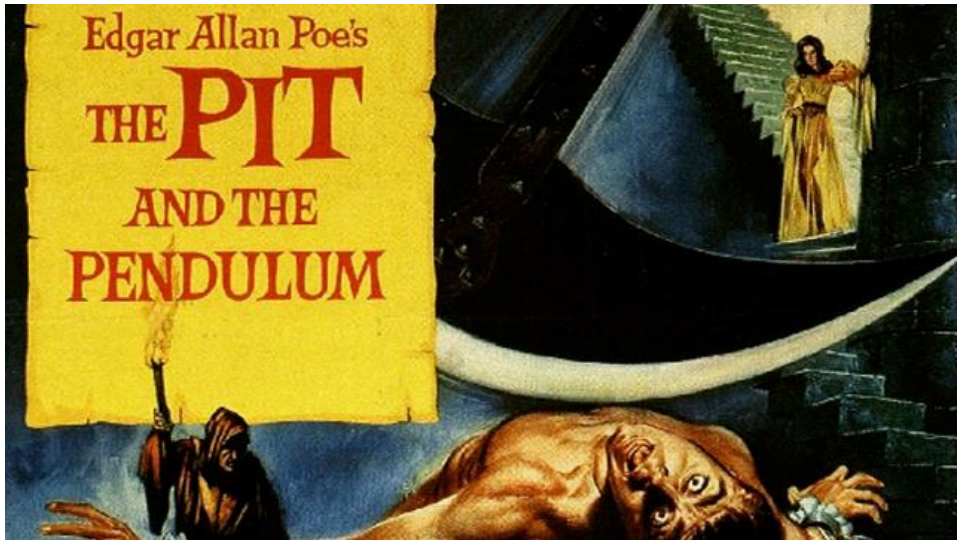
Physics Tutorial 7: Solvers



New Concepts

- Differentiating Constraints, Revisited
- Jacobian solutions
- The Constraint Force
- Global Solvers
- Linear Equations and Gauss-Seidel

Constraints, Revisted

Why do we need Constraints?

- “Things that need to be true, so our system will be accurate”
- That’s obvious in our approach to calculus, and our criticisms of it – our concern is the accuracy of the approach involved, stability, relation between complexity and time step, etc.
- Constraints turn this on its head

- As we said earlier this week, instead of asking what must be true in order for our system to be accurate, we’re asking what mustn’t be true in order for our system **not to be inaccurate**

Why do we need Constraints?

- A single object moving through space can happily be represented forever using numerical integration
- Once you have that object interact with things, integration itself isn't generally enough – things can stop the object moving, or affect how it moves.
- Constraints are a means of representing these interactions in concert with each other – instead of treating each object as an independent, perfect Newtonian entity, we establish what the object can and cannot do contextually.

Why do we need Constraints?

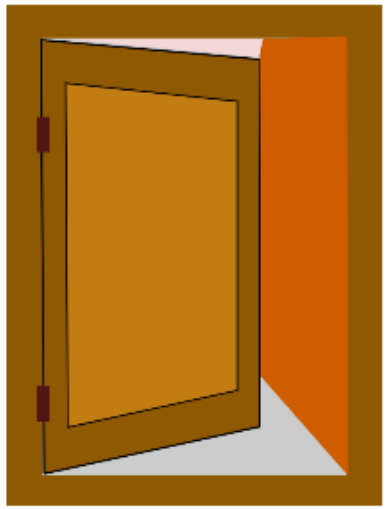
- Example: A Wheel

- How is this object constrained?
  - It can freely rotate about a single point (its axel) along a single axis
  - The wheel itself cannot move in linear fashion

Why do we need Constraints?

- Example: A Door on a Hinge

- How is this object constrained?
  - It can rotate about a single point (its hinge) along a single axis, but this rotation is bounded
  - The door cannot move in linear fashion

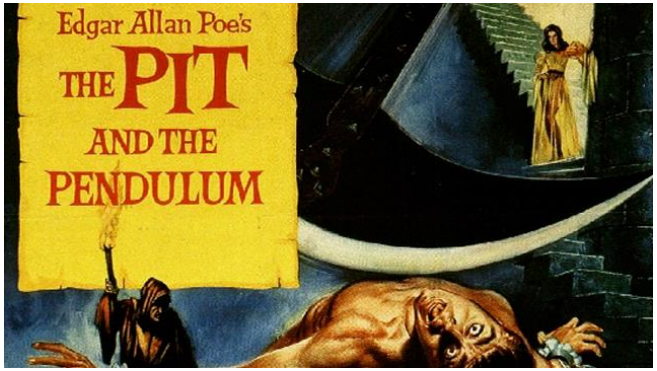


Why do we need Constraints?

- Example: A Book
- How is this object constrained?
  - Consider the cover and the pages to be different objects
  - Cover and the pages have a constraint between them, hinging them at the spine of the book
  - Opening the cover affects the pages

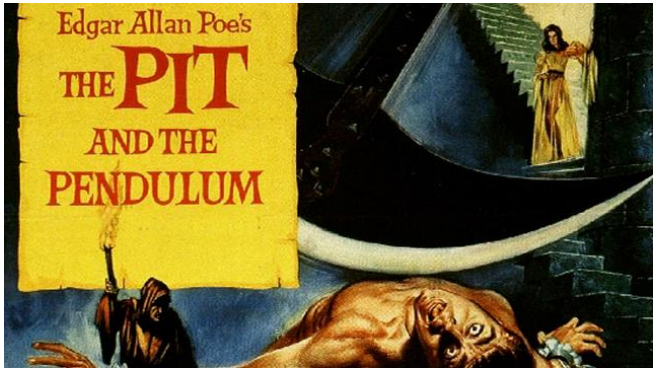
Why do we need Constraints?

- Example: A PENDULUM OF SWINGING DEATH
- So, what type of constraint is Poe's eponymous pendulum?



Why do we need Constraints?

- Example: A PENDULUM OF SWINGING DEATH
- So, what type of constraint is Poe's eponymous pendulum?
- Same as the door on a hinge – single constraint allowing bounded rotation about one axis



Resolving Constraints?

- In its simplest form, a Constraint system will take the form of a set of linear equations
- We pass in the velocities (as an example), both linear and angular, and the system resolves to a force to be applied to each object in the system
- Resolving these constraints is the purpose of a Solver, which we'll explore in-depth later in the lecture

Resolving Constraints?

- Constraint-based solvers permit us to resolve all constraints on a system simultaneously (e.g., in a single time-step). But why would we bother with resolving all constraints on all objects in one solver?
- What happens when we've got multiple constraints affecting the same object? A stack of boxes or a ball pool?
- By solving the system in its entirety, no one object will be affected more than the others (e.g., the box at the bottom of the stack won't be unduly pushed further than the other boxes)

The Jacobian

What is the Jacobian?

- Imagine two objects, A and B, with velocities $v_A v_b$ and angular velocities $\omega_A \omega_B$, respectively.
- A velocity constraint (representing the entire velocity of this system) needs to consider all four variables.
- Define that velocity constraint ***V*** as
- $$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v_A} \\ \boldsymbol{\omega_A} \\ \boldsymbol{v_B} \\ \boldsymbol{\omega_B} \end{bmatrix}$$

What is the Jacobian?

- By using the POWER OF MATH (see the calculus in the handout), we can calculate the rate at which the constraint changes value. This **shouldn't** be confused with rate of change of velocity (acceleration).

- If we differentiate the constraint C, to obtain $\dot{C}$ (pron. cee-dot) we get an equation of the form:
- $\dot{C} = j_1 \cdot v_A \ + \ j_2 \cdot \omega_A \ + \ j_3 \cdot v_B \ + \ j_4 \cdot \omega_B$
- $\boldsymbol{\dot{C} = JV = [j_1 \quad j_2 \quad j_3 \quad j_4] \begin{bmatrix} v_A \\ \omega_A \\ v_B \\ \omega_B \end{bmatrix} = 0}$

The Constraint Force

The Constraint Force

- So, the Jacobian's important – in fact, it's the key to unlocking constraint-based solvers

- We also need to consider the concept of *Work done* – this is a physical concept that comes, again, from Newtonian dynamics

- An object is considered to have done work W if it has moved a distance s under force F

$$W = F \cdot s$$

- Note that this is a dot product of two vectors
- Work done is a **scalar**

The Constraint Force

- Consider a weight of mass m resting on a table of height h. From previous lectures we know that there is a force (gravitational) pulling the weight downwards, and a countering force from the table which holds the weight in equilibrium
- In this system, no work is done, because s has no magnitude (the weight isn't going anywhere)
- If the table suddenly disappeared, the mass would accelerate at g (gravitational acceleration) and travel h metres towards the ground
- $F = mg$, ergo work done $W = mgh$ (which correlates to the weight's potential energy at height h)

The Constraint Force

- A constraint, as we said right at the beginning of the lecture, is something an item cannot do if our system is to avoid being inaccurate
- A constraint might stop an object from occupying a location (such as, the location of another object…)
- A constraint might ensure an object can only pivot about a certain axis, or a certain amount (like a knee joint)
- As such, a constraint **never adds energy to the system**

- This is a key concept – comes back to conservation of momentum and system stability

The Constraint Force

- So, let's consider the rate of work done (power P)
- Since we know that a constraint never adds energy (work) to the system, $P = 0$
- More fully: $P = F \cdot \frac{s}{\Delta t} = 0$
- We recall from yesterday that change in displacement over change in time is analogous to velocity v, ergo

$$F \cdot v = 0$$

The Constraint Force

- Considering again a two-body case, we can generalise this principle – each body has two quantities that keep them constrained (so four, total, in the system):

$$F = \begin{pmatrix} f_1 \\ \tau_1 \\ f_2 \\ \tau_2 \end{pmatrix}$$

- Remember that we're dealing with both linear and angular, hence our inclusion of torque
- You'll notice that this is of the form of our velocity constraint V; generalising this further, as constraint forces are perpendicular to the velocity vector, allows us to conclude that:

$$F \cdot V = 0$$

The Constraint Force

$$F \cdot V = 0$$

- This, really, is the basis for our constraint-based solver
- As this is the relationship our Jacobian has with the velocity vector V, we can use the Jacobian as the basis of our constraint force. By multiplying through by an unknown quantity λ, the constraint force becomes:

$$F = J^T \lambda$$

- Computing λ was discussed earlier in the week

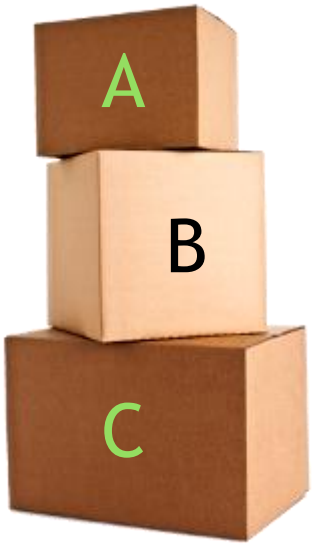
The Global Solver

The Global Solver

- So, by now we have (hopefully!) a good understanding of what a constraint is, and how constraints help us represent physical events in our simulation
- The global solver is the umbrella term for how we leverage constraints to handle an entire scene
- The rationale for it is fairly straightforward

The Global Solver

- Consider the example of a stack of boxes.
- If we resolve a constraint between B and C, that result will be invalidated by the resolution between A and B
- To avoid this we consider all constraints acting on B in the same set of linear equations

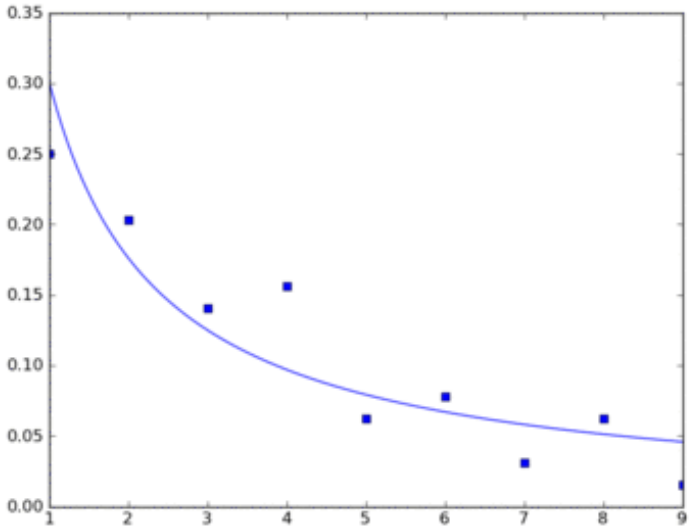


‘Global’ isn’t really ‘Global’

- But if you look at the way the system is operating we’re actually resolving constraints iteratively
- What this means is that our system is always moving towards the correct solution, even if it’s not there on any given frame

‘Global’ isn’t really ‘Global’

- All global solvers do this
- The important point is that all constraints are resolved as best they can, and more iterations move us towards greater accuracy (in the context of our linear equations – in the context of our time-stepped physics system, we tend away from accuracy)
- So how do we do it?



Solving Linear Equations

Consider the following equations…

$$4x + y = 23$$
$$x - z = 6$$
$$2z + y = 3$$

- Simple way of solving you'll have learned in school
- Rearrange to isolate a variable ($x = z + 6$, $y = 3 - 2z$)
- Substitute to compute z
- Rinse and repeat

Consider the following equations…

$$4x + y = 23$$
$$x - z = 6$$
$$2z + y = 3$$

- Straightforward approach to understand
- Not tractable for large systems – what if you have a dozen unknowns?
- Need a more computationally efficient approach

Matrix approach (*Ax* = *b* Form)

$$4x + y = 23$$
$$x - z = 6$$
$$2z + y = 3$$

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 23 \\ 6 \\ 3 \end{bmatrix}$$

- Rearrange our array of equations into a matrix
- Remembering how matrices multiply out, this should be intuitive
- Representation is scalable, i.e.:

Matrix approach (*Ax* = *b* Form)

$$\begin{bmatrix} a_{11} & a_{21} & \cdots & a_{i1} \\ a_{12} & a_{22} & \cdots & a_{i2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1j} & a_{2j} & \cdots & a_{ij} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \end{bmatrix}$$

$$\boldsymbol{A} \qquad \boldsymbol{x} \quad = \quad \boldsymbol{b}$$

- $\boldsymbol{A}$ is the Coefficient Matrix (the coefficients in our equation array, so the 2 in 2*x*)
- $\boldsymbol{x}$ is the Solution Vector – the unknowns we're attempting to compute
- $\boldsymbol{b}$ is the Constant Vector – the RHS of our equations
  - Also, Baumgarte – more on that later

Using linear solvers to resolve multiple constraints…

- Many approaches to solving the $\boldsymbol{Ax} = \boldsymbol{b}$ form

- **Jacobi Method**
  - Parallelisable (so good for GPU), but slow convergence
- **Gauss-Seidel**
  - Simplest to implement, default framework approach
- **Successive Over-Relaxation**
  - Faster convergence than Gauss-Seidel, more complex to implement
- **Conjugate Gradient Method**
  - More complex, faster convergence

- **And many, many more…**

Gauss-Seidel

- Iterates through each row of the A matrix
- Solves every constraint acting on a given object, in order
- Output is how the object's velocity must change in order to more closely satisfy its constraints

- Note the 'more closely'.
- It's an eventually consistent approach – always tending towards accuracy, but never quite reaching it so long as objects within the environment are interacting

Gauss-Seidel

- Assumes that all objects have non-zero mass (which must be true, because dividing by zero is bad).
- Unbounded solution time – convergence can take forever – imagine a wall with infinite mass, constrained to another wall with infinite mass – will take infinite time to compute
- Engineering solution: Iteration cap, will only go through a certain number of iterations before providing output
- Using last solution as a basis statistically improves performance (compared with starting from 0)

Gauss-Seidel

- Ordering of constraint solution MATTERS
- To converge towards same answer, each iteration must iterate over constraints in the same order every time
  - Imagine a stack of boxes.
  - First iteration, starts at the top, bottom box is most affected
  - Second iteration, starts at the bottom, top box is most affected
  - Never close to convergence because oscillates between priorities

Gauss-Seidel

- In implementation terms, just means the for-loop executes the same way each iteration
- Randomising the order of manifolds each time-step (NOT each iteration!) may lead to greater accuracy. Consider why…

Constraint Drift

Preventing Constraint Drift

- Time-series – errors spiral over time
- Problem made more apparent by eventually-consistent nature of solver
- Consider the distance constraint presented in a previous lecture
  - Constraint is based entirely on velocity
  - If one object moves in a single frame, constraint will not realise it, and continue to update based on new distance

Baumgarte Stabilisation

- Adds forces to the system to compensate for previous errors
- Can be considered $\boldsymbol{b}$ in our $\boldsymbol{Ax} = \boldsymbol{b}$
- In the framework, managed through addition of velocity based on length of constraint
- *Eventually* compensates for drift, in the same way that our solver is *eventually* consistent

Baumgarte Stabilisation

- Value for Baumgarte not readily predictable
- Will need fine-tuning
- Too little, doesn't prevent drift. Too much, system explodes.
- Usually a value between 0.1 and 0.3 that which would be needed to resolve the positional error in a single time-step is appropriate, but will need some experimentation

Summary

- Revisited Constraints
- Discussed the Jacobian
- Connected both back to force (on which our system *hinges*, pun intended)
- Introduced the concept of the global solver

- Linear Algebra
- Gauss-Seidel
- Baumgarte

Implementation

- Explore collision response
- Vary Baumgarte, experiment with it
- Stack objects
- Start coursework